Discovery of Band Order Dependencies

Li, Pei ; Szlichta, Jaroslaw ; Böhlen, Michael Hanspeter ; Srivastava, Dinesh

Abstract: We introduce band ODs to model the semantics ofattributes that are monotonically related with small variationswithout there being an intrinsic violation of semantics. Tomake band ODs relevant to real-world applications, we makethem less strict to holdapproximatelywith some exceptions andconditionallyon subsets of the data with a mix of ascendingand descending directions. Formulating integrity constraintsmanually requires domain expertise, is prone to human errors,and time consuming. Thus, we study the problem of automaticabcOD discovery. We propose an algorithm that utilizes thenotion of a longest monotonic band (LMB) to identify longestsubsequences of tuples that satisfy a band OD. We formulate theabcOD discovery problem as a constraint optimization problem,and devise a dynamic programming algorithm that determinesthe optimal solution in polynomial time (super-cubic complexity).To further optimize the performance over large datasets, weadapt the algorithm to consider pieces (contiguous sequencesof tuples) in a greedy fashion. This improves the performanceby orders-of-magnitude without sacrificing precision in practice.We show that for unidirectional abcODs, with only ascending ordescending orderings, our pieces-based algorithm is guaranteedto find the optimal solution. Finally, we perform a thoroughexperimental evaluation of our techniques over real-world andsynthetic datasets.

ABC of Order Dependencies

Pei Li ▽1, Jaroslaw Szlichta #2, Michael Böhlen $3, Divesh Srivastava ÷4

▽ Onedot A.G., Switzerland, # Ontario Tech University, Canada,
$ University of Zürich, Switzerland, ÷ AT&T Labs-Research, United States
1 pei.li@onedot.com, 2 jarek@uoit.ca, 3 boehlen@ifi.uzh.ch, 4 divesh@research.att.com

***Abstract*—We introduce band ODs to model the semantics of attributes that are monotonically related with small variations without there being an intrinsic violation of semantics. To make band ODs relevant to real-world applications, we make them less strict to hold *approximately* with some exceptions and *conditionally* on subsets of the data with a mix of ascending and descending directions. Formulating integrity constraints manually requires domain expertise, is prone to human errors, and time consuming. Thus, we study the problem of automatic abcOD discovery. We propose an algorithm that utilizes the notion of a longest monotonic band (LMB) to identify longest subsequences of tuples that satisfy a band OD. We formulate the abcOD discovery problem as a constraint optimization problem, and devise a dynamic programming algorithm that determines the optimal solution in polynomial time (super-cubic complexity). To further optimize the performance over large datasets, we adapt the algorithm to consider pieces (contiguous sequences of tuples) in a greedy fashion. This improves the performance by orders-of-magnitude without sacrificing precision in practice. We show that for unidirectional abcODs, with only ascending or descending orderings, our pieces-based algorithm is guaranteed to find the optimal solution. Finally, we perform a thorough experimental evaluation of our techniques over real-world and synthetic datasets.**

I. Introduction

A. Motivation

Modern data-intensive applications critically rely on high quality data to ensure that analyses are meaningful and do not fall prey to the garbage in, garbage out (GIGO) syndrome. In constraint-based data cleaning, dependencies are used to specify data quality requirements. Previous work has focused on functional dependencies (FDs) [1]. Several extensions to the notion of an FD have been studied, including *order dependencies* (ODs) [2], [3], [4], [5], [6], [7], which express rules involving order.

We introduce a novel data dependency *approximate band conditional OD* (abcOD). *Band ODs* express order relationships between attributes with small variations causing FDs and rules involving order including ODs [3], [6], [7], sequential dependencies [2] and denial constraints [8] to be violated without actual violation of application semantics. To match real world scenarios, we allow band ODs to hold *conditionally* over subsets of the data and *approximately* with some exceptions with a mix of *ascending* and *descending* order.

Table I contains 22 sample releases of the *Music* dataset (reprise records) from Discogs[1]. For tracking purposes music companies assign a catalog number (cat#) to each release of a particular label. When lexicographically ordered by attribute cat#, the release date (encoded using attributes year and month) is also approximately ordered over subsets of the data (with ascending and descending directions).

TABLE I
REPRISE RECORDS.

id	release	country	year	month	cat#
t_1	Unplugged	Canada	1992	Aug	CDW45024
t_2	Mirror Ball	Canada	**2012**	Jun	CDW45934
t_3	Ether	Canada	1996	Feb	CDW46012
t_4	Insomniac	Canada	1995	Oct	CDW46046
t_5	Summerteeth	Canada	1999	Mar	CDW47282
t_6	Sonic Jihad	Canada	2000	Jul	CDW47383
t_7	Title of...	Canada	1999	Jul	CDW47388
t_8	Reptile	Canada	2001	Mar	CDW47966
t_9	Always...	Canada	2002	Feb	CDW48016
t_{10}	Take A Picture	US	2000	Nov	9 16889-4
t_{11}	One Week	US	1998	Sep	9 17174-2
t_{12}	Only If...	US	1997	Nov	9 17266-2
t_{13}	Never...	US	1996	Nov	9 17503-2
t_{14}	We Run ...	US	1994	Dec	9 18069-2
t_{15}	The Jimi...	US	1982	Aug	9 22306-1
t_{16}	Never...	US	1987	Jan	9 25619-1
t_{17}	Vonda Shepard	US	1989	Mar	9 25718-2
t_{18}	Ancient Heart	US	**Null**	Jul	9 25839-2
t_{19}	Twenty 1	US	1991	May	9 26391-2
t_{20}	Stress	US	1990	Apr	9 26519-1
t_{21}	Play	US	1991	Mar	9 26644-2
t_{22}	Handels...	US	1992	Apr	9 26980-2

Release dates are approximately ordered within subsets of the tuples called *series*, i.e., $\{t_1\text{–}t_9\}, \{t_{10}\text{–}t_{14}\}$ and $\{t_{15}\text{–}t_{22}\}$. Note that tuple t_3 has a smaller cat# than t_4 (CDW46012 $<$ CDW46046), but is released a few months later than tuple t_4 (1996/Feb $>$ 1995/Oct; for month the sort order is according to the calendar ordering). This is common in the music industry as cat# is often assigned to a record before it is actually released at the production stage. Thus, tuples with delayed release dates will slightly violate an OD between cat# and (year, month). A permissible range to accommodate these small variations is called a *band*.

Attribute year has also a missing value (tuple t_{18}) and an erroneous value (tuple t_2) that severely break the OD between cat# and year, as the value of year for tuple t_2 is 2012 and for tuple t_3 is 1996 despite the ascending trend within the series. We verified that the correct value of year for tuple t_2 is 1995. (Table II shows statistics of violations.)

Similarly, since vehicle identification numbers (VIN) for cars are assigned sequentially, attributes VIN and year in car datasets are conditionally ordered over subsets of data. There are small variations to the OD between these attributes as VINs are assigned to a car before it is manufactured and year denotes

[1] www.discogs.com

TABLE II
STATISTICS OF TOP-5 MUSIC LABELS OF *Discogs*.

label	# total releases	# missing years	# incorrect years
Capitol Records	28935	3392	896
Reprise Records	9830	688	304
Ninja Tune	2055	10	33
V2 Records	1551	13	15
BGO Records	588	47	13

the time of the completion of the product. There are also actual errors to this OD (as illustrated in Figure 1a), due to data quality issues. Fig. 1 plots a small sample of the real-world *Car* dataset[2] and *Music* dataset series (separated by vertical lines). Series are identified by cat# and VIN, respectively.

Data dependencies to identify data quality errors can be obtained manually through a consultation with domain experts, but this is known to be an expensive, time consuming, and error-prone process [1], [2], [3], [6]. Thus, automatic approaches to discover data dependencies to identify data quality issues are needed. The key technical problem that we study is how to automatically discover abcODs.

B. Contributions

There are two variants of data dependency discovery algorithms. The first one is a global approach to automatically find all dependencies that hold in the data [1], [3], [6], [7]. The second one is a relativistic approach to find subsets of the data obeying the expected semantics [2], [9], which is laborious to do manually. We apply a *hybrid* approach to the discovery of abcODs that combines these two approaches.

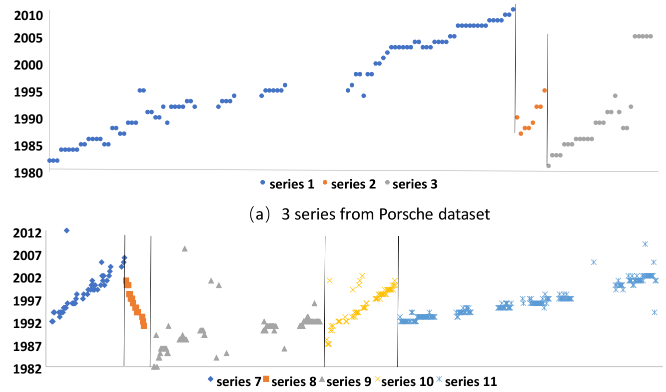
To automatically identify candidates for embedded band ODs without human intervention, we use a cheaper global approach that finds all traditional ODs within an approximation ratio [6], [7]. The approach in [6], [7] is limited to identifying ODs that (i) do not permit small variations within a band, thus, we deliberately set the approximation ratio higher, and (ii) hold over the entire dataset rather than subsets of the data, thus, we separate the data into segments by using a *divide-and-conquer* approach. We use the identified traditional ODs, ranked by the measure of interestingness [6], [7], as candidate embedded band ODs to solve the problem of discovering abcODs.

We define the problem of abcOD discovery as an *optimization problem* desiring parsimonious segments that identify large fractions of the data (*gain*) that satisfy the embedded band OD with few violations (*cost*).

We make the following contributions in this paper.

- We define a novel *band* OD integrity constraint based on small variations causing traditional ODs to be violated without an actual violation of application semantics. To make band ODs applicable to real-world data, we relax their requirements to hold *approximately* with some exceptions and *conditionally* on subsets of the data. We develop a method to automatically compute the *band-width* to allow for small variations.

[2]www.classicdriver.com



(a) 3 series from Porsche dataset

(b) 5 series from Reprise Records

Fig. 1. Real-world series in *Car* and *Music* datasets.

- We formulate the *abcOD discovery problem* as a constraint optimization problem. The naive solution to consider all possible segmentations of tuples is prohibitively expensive, as it leads to exponential time complexity. Thus, we devise discovery algorithms based on the proposed notion of *longest monotonic bands* (LMBs) to identity longest subsequences of tuples that satisfy a band OD. We devise a *dynamic programming* algorithm based on LMBs that finds the optimal solution in polynomial time (super-cubic complexity). To further decrease the search space over large datasets, we propose a greedy algorithm based on *pieces*, which are contiguous monotonic sequences of tuples. Our greedy algorithm is orders-of-magnitude faster than the optimal algorithm without sacrificing the precision in practice. When bidirectionality is removed to consider *unidirectional* abcODs, we show that the pieces-based algorithm finds the optimal solution.
- We experimentally demonstrate the effectiveness and scalability of our solution, and compare our techniques with baseline methods on real-world and syntethic datasets.

We provide basic definitions in Sec. II. Sec. III introduces the concept of LMBs. In Sec. IV and V, we study algorithms to discover abcODs. We discuss experimental results in Sec. VI and related work in Sec. VII. We conclude in Sec. VIII.

II. BACKGROUND

We use the following notational conventions.

- **Relations.** R denotes a relation schema and r denotes a specific table instance. Italic letters from the beginning of the alphabet A, B and C denote single attributes. Also, s and t denote tuples in r and $s.A$ denotes the value of an attribute A in a tuple s. $dom(A)$ denotes the domain of an attribute A.
- **Lists.** Bold letters from the end of the alphabet **X, Y** and **Z** denote lists of attributes. $[A, B, C]$ denotes an explicit list of attributes. $dom(\mathbf{X}) = dom(A) \cdot dom(B) \cdot dom(C)$ denotes the domain of $\mathbf{X}$, where $\mathbf{X} = [A, B, C]$. $s.\mathbf{X}$ denotes the value of the list of attributes $\mathbf{X}$ in the tuple s.

Let $d : dom(\mathbf{X}) \cdot dom(\mathbf{X}) \rightarrow \mathbb{R}$ be a *distance function* defined on the domain of $\mathbf{X}$. Distance function d satisfies the following properties: *anti-symmetry, triangle inequality* and *identity of indiscernibles*. We consider $d(x_1,x_2) = ||x_2|| - ||x_1||$, where $||x||$ denotes the norm of the value list x.

We model an *order specification* as a directive to sort a dataset in *ascending* or *descending* order.

Definition 2.1 (Order Specification): An order specification is a list of marked attributes, denoted as $\overline{\mathbf{Y}}$. There are two ordering directions: asc and desc, indicating ascending and descending ordering, respectively. As shorthand, $\mathbf{Y}\uparrow$ indicates $\mathbf{Y}$ asc and $\mathbf{Y}\downarrow$ indicates $\mathbf{Y}$ desc. □

Definition 2.2 (Operator $\preceq_{\Delta,\overline{\mathbf{Y}}}$): Let $\overline{\mathbf{Y}}$ be a list of marked attributes in a relation r and let Δ be a constant value. For two tuples $t,s \in r$, $t \preceq_{\Delta,\overline{\mathbf{Y}}} s$ if

- $\overline{\mathbf{Y}} = \mathbf{Y}\uparrow$ and $d(s.\mathbf{Y},t.\mathbf{Y}) \leq \Delta$; or
- $\overline{\mathbf{Y}} = \mathbf{Y}\downarrow$ and $d(s.\mathbf{Y},t.\mathbf{Y}) \geq -\Delta$.

Let $t \preceq_{\overline{\mathbf{Y}}} s$ be the operator $t \preceq_{\Delta,\overline{\mathbf{Y}}} s$, where $\Delta = 0$. □

Definition 2.3 (Band Order Dependency): Given a band-width Δ, a list of attributes $\mathbf{X}$, a list of marked attributes $\overline{\mathbf{Y}}$ over a relation schema R, a *band order dependency* (band OD) denoted by $\mathbf{X} \mapsto_\Delta \overline{\mathbf{Y}}$ holds over a table r if $t \preceq_{\mathbf{X}\uparrow} s$ implies $t \preceq_{\Delta,\overline{\mathbf{Y}}} s$ for every tuple pair $t,s \in r$. □

Example 2.4: A band OD cat# $\mapsto_{\Delta=1}$ year$\uparrow$ holds over tuples $\{t_1,t_3–t_9\}$ in Table I with a band-width of one year. A band OD cat# $\mapsto_{\Delta=12}$ [year, month]$\uparrow$ holds over tuples $\{t_1,t_3-t_9\}$ in Table I with a band-width of 12 months. □

Band ODs specify that when tuples are ordered increasingly on antecedent (left-hand-side; cat# in Example 2.4), their consequent (right-hand-side; year in Example 2.4) must be ordered non-decreasingly (e.g., wrt series S_1 and S_3 in Example 2.5 discussed next) or non-increasingly (e.g., wrt the series S_2 in Example 2.5) within the specified band-width (e.g., $\Delta = 1$ in Example 2.4). In real-world applications, band ODs often hold *approximately* with some exceptions to accommodate errors and *conditionally* over subsets of the data (*series*).

Example 2.5: There are three series in Table I: $S_1 = \{t_1 - t_9\}$ wrt cat# $\mapsto_{\Delta=1}$ year$\uparrow$, $S_2 = \{t_{10} - t_{14}\}$ wrt cat# $\mapsto_{\Delta=1}$ year$\downarrow$ and $S_3 = \{t_{15} - t_{22}\}$ wrt cat# $\mapsto_{\Delta=1}$ year$\uparrow$. There is a tuple with an erroneous year (t_2; correct year is 1995), and a tuple with a missing year (t_{18}; correct year is 1988). □

Since both ascending and descending trends are allowed, the consequent of the dependency is a list of marked attributes. We describe how to automatically compute band-width in Sec. III-D. Note that traditional ODs [3], [4], [5], [6], [7] are a special case of band ODs, where $\Delta = 0$. We support other data types than numerical columns including categorical columns. For instance, months can be represented as strings as in the example in Section I-A over Table I. Whenever the distance function can be preserved, the values of the columns are replaced with integers: 1, ..., n, in a way that keeps the same ordering, i.e., higher values are replaced by larger integers. Computation over integers is more time and space efficient.

We desire parsimonious series that identify large subsets of data that satisfy a band OD with few violations. We formally define the *approximate band conditional OD (abcOD) discovery problem* as an optimization problem in Sec. IV.

III. Longest Monotonic Bands

A naive solution to the abcOD discovery problem is to consider all possible segmentations 2^{n-1} of n tuples over the dataset. In order to reduce the search space, we introduce the notion of a *longest monotonic band* (LMB) to identify the longest subsequences of tuples that satisfy a band OD. We formally define LMBs in Sec. III-A, present how to efficiently calculate LMBs in Sec. III-B with computation details in Sec. III-C. We use LMBs in Sec. III-D to automatically compute band-width. (Thus, Sec. III.D is presented after Sections III.A–C). The automatically computed band-width is used as an input to the discovery algorithm of abcODs in Section IV.

A. Defining LMB

In contrast to previous methods [2], [3], [6], our definition of longest monotonic bands allows for *slight variations*. Recall, that when we consider a band OD cat# $\mapsto_{\Delta=1}$ year$\uparrow$ and a series $S_1 = \{t_1 - t_9\}$ in Figure 2 over Table I, tuples t_4 and t_7 are correct and only tuple t_2 is incorrect. We define LMBs with respect to a band OD $\mathbf{X} \mapsto_\Delta \overline{\mathbf{Y}}$. In the remaining, $T = \{t_1,t_2,\cdots,t_n\}$ denotes a sequence of tuples ordered lexicographically by $\mathbf{X}$ in ascending order.

Definition 3.1 (Longest Monotonic Band): Given a sequence of tuples $T = \{t_1,t_2,\cdots,t_n\}$, list of marked attributes $\overline{\mathbf{Y}}$ and band-width Δ, a *monotonic band* (MB) is a subsequence of tuples $M = \{t_i,\cdots,t_j\}$ over T, such that $\forall_{k_1,k_2 \in \{i,\cdots,j\},k_1<k_2}$ $t_{k_1} \preceq_{\Delta,\overline{\mathbf{Y}}} t_{k_2}$, where $\overline{\mathbf{Y}} = \mathbf{Y}\uparrow$ or $\mathbf{Y}\downarrow$. The longest subsequence M satisfying this condition over T is called a *longest monotonic band* (LMB). A sequence M is called an *increasing band* (IB) (and a *longest IB* (LIB) if M is a LMB) if $\overline{\mathbf{Y}} = \mathbf{Y}\uparrow$ and a *decreasing band* (DB) (and a *longest DB* (LDB) if M is a LMB) if $\overline{\mathbf{Y}} = \mathbf{Y}\downarrow$. □

Example 3.2: Consider band OD cat# $\mapsto_{\Delta=1}$ $\overline{\text{year}}$ over Table I ordered by cat#. Suppose tuples $T = \{t_{10} - t_{14}\}$ form one series. There is a LDB $\{t_{10} - t_{14}\}$ (with year $\{'00,'98,'97,'96,'94\}$) in T and there are two LIBs $\{t_{11},t_{12}\}$ ($t_{11} \preceq_{\Delta=1,\text{year}\uparrow} t_{12}$ holds with year $\{'98,'97\}$) and $\{t_{12},t_{13}\}$ ($t_{12} \preceq_{\Delta=1,\text{year}\uparrow} t_{13}$ holds with year $\{'97,'96\}$) in T. Thus, a LMB over T is $\{t_{10} - t_{14}\}$. Note that a LIB can be obtained with local decreases (and analogously LDB with local increases) within the band-width. Based on Definition 3.1, since $t_{10} \preceq_{\Delta=1,\text{year}\uparrow} t_{11}$, $t_{11} \preceq_{\Delta=1,\text{year}\uparrow} t_{13}$ and $t_{11} \preceq_{\Delta=1,\text{year}\uparrow} t_{14}$ do not hold, tuples t_{10}, t_{13} and t_{14} are not part of the LIB with tuples t_{11} and t_{12}. However, $t_{10} \preceq_{\Delta=1,\text{year}\downarrow} t_{11}$, $t_{10} \preceq_{\Delta=1,\text{year}\downarrow} t_{12}$, $t_{10} \preceq_{\Delta=1,\text{year}\downarrow} t_{13}$, $t_{10} \preceq_{\Delta=1,\text{year}\downarrow} t_{14}$, $t_{11} \preceq_{\Delta=1,\text{year}\downarrow} t_{12}$, $t_{11} \preceq_{\Delta=1,\text{year}\downarrow} t_{13}$, $t_{11} \preceq_{\Delta=1,\text{year}\downarrow} t_{14}$, $t_{12} \preceq_{\Delta=1,\text{year}\downarrow} t_{13}$, $t_{12} \preceq_{\Delta=1,\text{year}\downarrow} t_{14}$, $t_{13} \preceq_{\Delta=1,\text{year}\downarrow} t_{14}$. Thus, tuples t_{10}–t_{14} form a LDB. LMBs are *not* necessarily contiguous subsequences of tuples. For example, in Fig. 2 a LIB over series with tuples t_1–t_9 includes tuples $t_1 \cup t_3$–t_9. □

Definition 3.3 (Maximal & Minimal Tuples): Given a sequence of tuples $T = \{t_1,\cdots,t_n\}$ and a list of attributes $\mathbf{Y}$, a tuple $t_i \in T$ is a *maximal tuple*, denoted as $\max_{\mathbf{Y}}(t_1,\cdots,t_n)$,

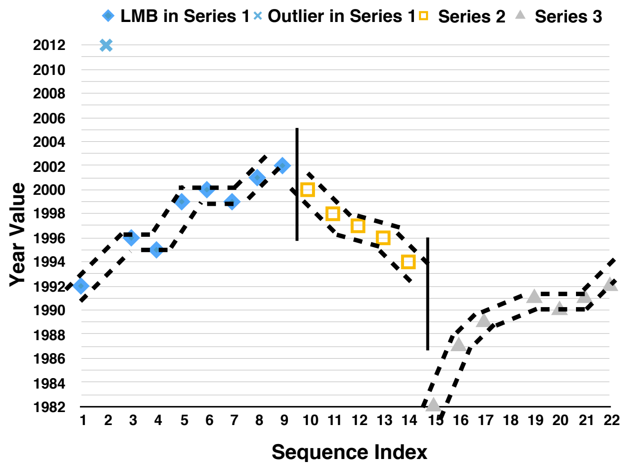


Fig. 2. Determining abcODs in Table I, Δ=1.

i	0	1	2	3	4	5	6	7	8	9
k		'92	'12	'96	'95	'99	'00	'99	'01	'02
0	0	0	0	0	0	0	0	0	0	0
	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	'92	'92	'92	'92	'92	'92	'92	'92	'92
	0	'92	'12	'12	'12	'12	'12	'12	'12	'12
2	∞	∞	'12	'96	'96	'96	'96	'96	'96	'96
	0	0	0	'96	'96	'99	'00	'00	'01	'02
3	∞	∞	∞	∞	'96	'96	'96	'96	'96	'96
	0	0	0	0	'95	'95	'99	'99	'00	'01
4	∞	∞	∞	∞	∞	'99	'99	'99	'99	'99
	0	0	0	0	0	0	0	'99	'99	'99
5	∞	∞	∞	∞	∞	∞	'00	'99	'99	'99
	0	0	0	0	0	0	0	0	0	0
6	∞	∞	∞	∞	∞	∞	∞	'00	'00	'00
	0	0	0	0	0	0	0	0	0	0
7	∞	∞	∞	∞	∞	∞	∞	∞	'01	'01
	0	0	0	0	0	0	0	0	0	0
8	∞	∞	∞	∞	∞	∞	∞	∞	∞	'02
	0	0	0	0	0	0	0	0	0	0
9	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	0	0	0	0	0	0	0	0	0	0

Fig. 3. Matrix of best tuples of MBs.

if $\forall_{j\in\{1,\cdots,n\}}\ d(t_j.\mathbf{Y}, t_i.\mathbf{Y}) \geq 0$ and a *minimal tuple* denoted as $\min_{\mathbf{Y}}(t_1,\cdots,t_n)$ if $\forall_{j\in\{1,\cdots,n\}}\ d(t_j.\mathbf{Y}, t_i.\mathbf{Y}) \leq 0$. □

Example 3.4: Given $T = \{t_3–t_7\}$ over Table I, t_4 is a minimal tuple and t_6 is a maximal tuple wrt year. □

B. Calculating LMBs

We propose an efficient approach to calculate LMBs by reducing the problem of finding a LMB in a sequence of tuples to the sub-problem of finding monotonic bands of all possible lengths. Once MBs are enumerated, the longest one is picked as a LMB. $T[i]$ denotes the *prefix* of a sequence T of length i, i.e., $T[i] = \{t_1, t_2, \cdots, t_i\}$, where $i \in \{0,\cdots,n-1\}$ and $T[0] = \emptyset$. The following theorem leads to an effective solution of calculating LMBs. The proofs of all theorems and lemmas can be found in Appendix.

Theorem 1: Given a band-width Δ, a sequence of tuples T and a list of attributes $\mathbf{Y}$, let $\mathsf{IB}_{k,i}$ denote an IB with the smallest maximal tuple $s_{k,i}$ among all IBs with length k in a prefix $T[i]$.

1) If $d(t_{i+1}.\mathbf{Y}, s_{k,i}.\mathbf{Y}) \leq \Delta$, then there are two candidates for $\mathsf{IB}_{k+1,i+1}$: $\mathsf{IB}_{k+1,i}$ with $s_{k+1,i}$ being its maximal tuple; and a new IB $= \mathsf{IB}_{k,i} \cup \{t_{i+1}\}$ with its maximal tuple over $s_{k,i}$ and t_{i+1}. If $s_{k+1,i}$ is a maximal tuple among $\{s_{k,i}, s_{k+1,i}, t_{i+1}\}$, then $\mathsf{IB}_{k+1,i+1} = \mathsf{IB}_{k,i} \cup \{t_{i+1}\}$ and $s_{k+1,i+1} = \max_{\mathbf{Y}}(s_{k,i}, t_{i+1})$.
2) Else, $\mathsf{IB}_{k+1,i+1} = \mathsf{IB}_{k+1,i}$ and $s_{k+1,i+1} = s_{k+1,i}$ □

An analogous result holds for decreasing bands. Based on Theorem 1 and its analog for decreasing bands, to find a LMB in a sequence of tuples, it is sufficient to maintain two tuples for each possible $k, i \in \{0,\cdots,n-1\}$: (1) the smallest maximal tuple of IBs of length $k+1$ in a prefix $T[i+1]$, and (2) the largest minimal tuple of DBs of length $k+1$ in $T[i+1]$.

Definition 3.5 (Best tuples): Given a sequence of tuples T, band-width Δ and a list of attributes $\mathbf{Y}$, for each $i, k \in \{1,\cdots n\}$, $(s_{k,i}, l_{k,i})$ are the *best tuples* of MBs of length k in $T[i]$ if (1) $s_{k,i}$ is the smallest maximal tuple of an IB with length k in $T[i]$, and (2) $l_{k,i}$ is the largest minimal tuple of a DB with length k in a prefix $T[i]$. Let initially $(s_{0,i}.\mathbf{Y}, l_{0,i}.\mathbf{Y})$ equal $(\{0,0,\cdots,0\},\{\infty,\infty,\cdots,\infty\})$ for $i \in \{0,\cdots,n\}$ and $(s_{k,0}.\mathbf{Y}, l_{k,0}.\mathbf{Y})$ equal to $(\{\infty,\infty,\cdots,\infty\},\{0,0,\cdots,0\})$ for $k \in \{1,\cdots,n\}$. The best tuples $(s_{k,i}, l_{k,i})$ of monotonic band with length k in a prefix $T[i]$ satisfy the following recurrence, where $u = \min_{\mathbf{Y}}(s_{k+1,i}, \max_{\mathbf{Y}}(t_{i+1}, s_{k,i}))$ and $v = \max_{\mathbf{Y}}(l_{k+1,i}, \min_{\mathbf{Y}}(t_{i+1}, l_{k,i}))$. For simplicity, tuples are represented by their $\mathbf{Y}$ values in Example 3.6.

$$(s_{k+1,i+1}, l_{k+1,i+1}) =
\begin{cases}
(u, v) & d(t_{i+1}.\mathbf{Y}, s_{k,i}.\mathbf{Y}) \leq \Delta \;\&\; d(t_{i+1}.\mathbf{Y}, l_{k,i}.\mathbf{Y}) \geq -\Delta \\
(s_{k+1,i}, v) & d(t_{i+1}.\mathbf{Y}, s_{k,i}.\mathbf{Y}) > \Delta \;\&\; d(t_{i+1}.\mathbf{Y}, l_{k,i}.\mathbf{Y}) \geq -\Delta \\
(u, l_{k+1,i}) & d(t_{i+1}.\mathbf{Y}, s_{k,i}.\mathbf{Y}) \leq \Delta \;\&\; d(t_{i+1}.\mathbf{Y}, l_{k,i}.\mathbf{Y}) < -\Delta \\
(s_{k+1,i}, l_{k+1,i}) & d(t_{i+1}.\mathbf{Y}, s_{k,i}.\mathbf{Y}) > \Delta \;\&\; d(t_{i+1}.\mathbf{Y}, l_{k,i}.\mathbf{Y}) < -\Delta
\end{cases}$$

□

Example 3.6: Consider a sequence $T = \{'92, '96, '95\}$ and band-width $\Delta = 1$. There are three IBs of length 1: $\{'92\}, \{'96\}$, and $\{'95\}$ in T, among which $'92$ is the smallest maximal tuple. Accordingly, there are also the three same DBs of length 1, where $'96$ is the largest minimal tuple. Thus, $('92, '96)$ are the best tuples of MBs with length 1. Similarly, there is one DB with length 2: $\{'96, '95\}$, where $'95$ is the largest minimal tuple. There are three IBs with length 2: $\{'92, '96\}, \{'96, '95\}$ and $\{'92, '95\}$, among which $'95$ is the smallest maximal value. Thus, $('95, '95)$ are the best tuples of MBs of length 2 in T. □

Based on the recurrence in Def. 3.5, best tuples for monotonic bands can be computed recursively. Assume for each $k \in \{1,\cdots,n\}$ the best tuples $(s_{k,i}, l_{k,i})$ for MBs with length k in a prefix $T[i]$ are found. If $d(t_{i+1}.\mathbf{Y}, s_{k,l}.\mathbf{Y}) \leq \Delta$ holds, then a new IB of length $k+1$ is found, where the maximal tuple is $\max_{\mathbf{Y}}(t_{i+1}, s_{k,i})$. Thus, the smallest maximal tuple $s_{k+1,i+1}$ is chosen between $s_{k+1,i}$ and $\max_{\mathbf{Y}}(t_{i+1}, s_{k,i})$, i.e., $s_{k+1,i+1} = \min_{\mathbf{Y}}(s_{k+1,i}, \max_{\mathbf{Y}}(t_{i+1}, s_{k,i}))$. Otherwise, the smallest maximal tuple among IBs with length $k+1$ remains unchanged, i.e., $s_{k+1,i+1} = s_{k+1,i}$. DBs are computed analogously.

Example 3.7: Assume $T = \{t_1 - t_9\}$ over Table I, $\mathbf{Y} = [\text{year}]$ and $\Delta = 1$. Fig. 3 presents a matrix, where the $(k,i)^{th}$ entry

$\binom{s_{k,i}.\mathbf{Y}}{l_{k,i}.\mathbf{Y}}$ denotes $\mathbf{Y}$ value of the smallest maximal tuple $s_{k,i}$ for IBs with length k, and that of the largest minimal tuple $l_{k,i}$ for DBs with length k in $T[i]$. Initially, $(s_{0,i}.\mathbf{Y}, l_{0,i}.\mathbf{Y})$ is set to $(0,\infty)$ and $(s_{k,0}.\mathbf{Y}, l_{k,0}.\mathbf{Y})$ to $(\infty,0)$ for $i \in \{0,\cdots,9\}$ and $k \in \{1,\cdots,9\}$. We first test if t_1 (with year $'92$) can extend any MB, i.e., $(s_{k,0}, l_{k,0})$, $k \in [0,9]$. Since $d('92, s_{0,0}.\mathbf{Y}) < \Delta$, a new IB with length 1 with a maximal tuple t_1 is found. Similarly, since $d('92, l_{0,0}.\mathbf{Y}) > -\Delta$, a new DB with length 1 with a minimal tuple t_1 is found. We set $(s_{1,1}, l_{1,1})$ to $(\max_{\mathbf{Y}}(t_1, s_{0,0}), \min_{\mathbf{Y}}(t_1, l_{0,0})) = (t_1, t_1)$ (represented by year $('92,'92)$). For each $k \in \{1,\cdots,8\}$, $d('92, l_{k,0}.\mathbf{Y}) < -\Delta$ and $d('92, s_{k,0}.\mathbf{Y}) > \Delta$. Thus, $(s_{k+1,1}, l_{k+1,1})$ is set to $(s_{k+1,0}, l_{k+1,0})$. $\{t_2\text{-}t_9\}$ are processed accordingly with results of best tuples in Fig. 3. □

C. Computation Details

To find a LMB in a sequence of tuples T two arrays of size n are used to store the best tuples of MBs. Algorithm 1 presents the pseudo-code for computing a LMB. Arrays B_{inc} and B_{dec} store the best tuples (s_k, l_k) for each $k \in \{1,\cdots,n\}$, i.e., $B_{\mathsf{inc}}[k] = s_k, B_{\mathsf{dec}}[k] = l_k$. For each element t_i in T, B_{inc} and B_{dec} are updated by finding the best positions of t_i in B_{inc} and B_{dec}, denoted by k_1 to k_4, as follows (Line 6–Line 7).

- k_1 is the smallest index in B_{inc} that satisfies $d(t_i.\mathbf{Y}, s_{k_1}.\mathbf{Y}) > 0$. It is the shortest length of IBs with a smallest maximal tuple that ends at t_i in $T[i]$.
- k_2 is the smallest index in B_{inc} that satisfies $d(t_i.\mathbf{Y}, s_{k_2}.\mathbf{Y}) > \Delta$. It is the longest length of IBs with a smallest maximal tuple that ends at t_i in $T[i]$.
- k_3 is the smallest index in B_{dec} that satisfies $d(t_i.\mathbf{Y}, l_{k_3}.\mathbf{Y}) < 0$. It is the shortest length of DBs with a largest minimal tuple that ends at t_i in $T[i]$.
- k_4 is the smallest index in B_{dec} that satisfies $d(t_i.\mathbf{Y}, l_{k_4}.\mathbf{Y}) < -\Delta$. It is the longest length of DBs with a largest minimal tuple that ends at t_i in $T[i]$.

P_{inc} and P_{dec} are two arrays of size n that store the set of lengths of MBs with best tuples ending at t_i for each $i \in \{1,\cdots,n\}$, i.e., $P_{\mathsf{inc}}[i] = \{k \mid k \in \{k_1,\cdots,k_2\}\}$ and $P_{\mathsf{dec}}[i] = \{k \mid k \in \{k_3,\cdots,k_4\}\}$. For each $k \in \{k_1,\cdots,k_2\}$, $B_{\mathsf{inc}}[k]$ is updated by $\max_{\mathbf{Y}}(s_{k-1}, t_i)$ and adding k to $P_{\mathsf{inc}}[i]$ (Line 12); and for each $k \in \{k_3,\cdots,k_4\}$, $B_{\mathsf{dec}}[k]$ is updated by $\min_{\mathbf{Y}}(l_{k-1}, t_i)$ and adding k to $P_{\mathsf{dec}}[i]$ (Line 16).

Example 3.8: Assume $T = \{t_1 - t_9\}$ over Table I, $\mathbf{Y} = [\text{year}]$ and $\Delta = 1$. Initially, $B_{\mathsf{inc}}[i] = t_\infty$ with year ∞ and $B_{\mathsf{dec}}[i] = t_0$ with year 0 for each $i \in \{1,\cdots,9\}$ and P_{inc} and P_{dec} are empty. We start with t_1 with year $'92$. k_1 (and k_2, respectively) is computed by finding the positions of t_1 in array B_{inc}, so that $B[k_1]$ ($B[k_2]$) is the first-left tuple in B_{inc} whose year is greater than $'92$ ($'92+\Delta$). In both cases, $k_1 = k_2 = 1$, thus, $B_{\mathsf{inc}}[1]$ is replaced by t_1, and $k_1 = k_2 = 1$ is inserted into $P_{\mathsf{inc}}[1]$. Next, t_2 with year $'12$ is considered. With the updated array B_{inc}, $k_1 = k_2 = 2$ is inserted into $P_{\mathsf{inc}}[2]$ and $B_{\mathsf{inc}}[2] = t_2$ is set. The remaining tuples are processed accordingly with results reported in Figure 4. □

Next, we describe how to compute a LMB over T given the best tuple matrix stored in P_{inc} and P_{dec}. The path of a LIB is constructed in a sequence of tuples T in reverse order by scanning the array P_{inc}. Let $k \in P_{\mathsf{inc}}[i_k]$ be the largest value in P_{inc}, i.e., there exists a LMB of length k in T; and t_{i_k} is found as the k^{th} tuple in the LIB. Starting from $P_{\mathsf{inc}}[i_k]$ and k, P_{inc} is scanned in reverse order until the first tuple $P_{\mathsf{inc}}[i_{k-1}]$ is found that contains $k-1$. Then, $t_{i_{k-1}}$ is found, the $k-1^{th}$ tuple in the LMB. P_{inc} is continued to be scanned until all k tuples in the LIB are found (Line 17). A LDB is computed accordingly (Line 18). A LMB is chosen as the longest between a LIB and a LDB.

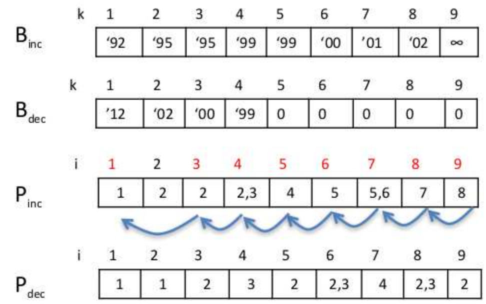


Fig. 4. Finding LMB; tuples in B_{inc} (B_{dec}) are represented by year.

Algorithm 1: Computing LMB

input : $T = \{t_1, t_2, \cdots, t_n\}$, band width Δ
output : LMB in T

```
1  for i ← 1 to n do
2      t∞.Y ← ∞; t0.Y ← 0; B_inc[i] ← t∞; B_dec[i] ← t0
3      P_inc[i] ← ∅; P_dec[i] ← ∅
4  k_inc = 0; k_dec = 0
5  for i ← 1 to n do
6      k1 ← posInc(B_inc, t_i, 0); k2 ← posInc(B_inc, t_i, Δ)
7      k3 ← posDec(B_dec, t_i, 0); k4 ← posDec(B_dec, t_i, −Δ)
8      k_inc ← max_Y(k_inc, k2); k_dec ← max_Y(k_dec, k4)
9      for k ← k2 to k1 do
10         b ← t0
11         if k > 1 then b ← B_inc[k − 1]
12         B_inc[k] ← max_Y(b, t_i); append(P_inc[i], k)
13     for k ← k4 to k3 do
14         b ← t∞
15         if k > 1 then b ← B_dec[k − 1]
16         B_dec[k] ← min_Y(b, t_i); append(P_dec[i], k)
17 if k_inc ≥ k_dec then L ← band(P_inc, k_inc)
18 else L ← band(P_dec, k_dec)
19 return L
```

Example 3.9: Consider $T = \{t_1 - t_9\}$ over Table I, $\mathbf{Y} = [\text{year}]$ and $\Delta = 1$. Fig. 4 shows the arrays P_{inc} and P_{dec} for finding a LIB and a LDB, respectively. To find a LIB P_{inc} is scanned to find the largest value 8 in $P_{\mathsf{inc}}[9]$. Thus a LIB with length 8 exists in T and t_9 is its eighth tuple. By a reverse scan (marked with arrows in Fig 4) from $P_{\mathsf{inc}}[9]$, the 7-th tuple t_8 is found. The operation is continued until all tuples in a LIB are found; i.e., $\{t_1('92), t_3('96), t_4('95), t_5('99), t_6('00), t_7('99), t_8('01), t_9('02)\}$. Since the length of a LDB over T found in a similar fashion is $4 < 8$, a LIB becomes a LMB. □

To find LMBs in the sequence T, best tuples are the key. Since a tuple $B_{\mathsf{inc}}[k_1]$ is updated by the algorithm by

$\max(s_{k_1-1}.\mathbf{Y}, t_i.\mathbf{Y})$, where $d(t_i.\mathbf{Y}, s_{k_1-1}.\mathbf{Y}) > 0$, the corresponding band $\mathsf{IB}_{\mathsf{k_1,i}}$ is an IB with smallest maximal tuples that ends at tuple t_i in the sequence $T[i]$. It is also a monotonic band with the shortest length, as k_1 is the smallest index in B_{inc}. Similarly, $\mathsf{IB}_{\mathsf{k_2,i}}$ is an IB of the longest length among IBs with the smallest maximal tuple that ends at t_i in $T[i]$. For each $t_i \in T$, the lengths of IBs with the smallest maximal tuples that end at t_i fall into range $[k_1, k_2]$. The length of a LIB in $T[i]$ is the maximal value in array $P_{\mathsf{inc}}[i]$. Accordingly, Alg. 1 finds a LDB with the largest minimal tuple in T. For each tuple in T of size n, it takes $O(\log n)$ time to update arrays B_{inc}, B_{dec}, P_{inc} and P_{dec}. Thus, Alg. 1 takes $O(n \log n)$ time to find a LMB in T . Each tuple t_i inserts maximally $\Delta+1$ values into arrays P_{inc} and P_{dec}. Thus, Alg. 1 takes $O((\Delta+1)n)$ space.

Theorem 2: Alg. 1 correctly finds a LMB in a sequence of tuples T of size n in $O(n \log n)$ time and $O((\Delta+1)n)$ space.□

D. Automatic Band-Width Estimation

Our goal is to effectively identify outliers in a sequence of tuples, while being tolerant to tuples that slightly violate an OD. Since band ODs hold over subsets of data called series (with ascending and descending trends), to identify the correct band-width, we separate the entire sequence of tuples T (ordered by $\mathbf{X}$) over a table r into contiguous subsequences of tuples S. We identify contiguous subsequences of tuples by using divide-and-conquer method, such that tuples in S satisfy a traditional OD $\mathbf{X} \mapsto \mathbf{Y}$ within approximation ratio. (Details of how to generate candidate abcODs based on global approach to find traditional ODs [6], [7] are in Sec. IV-A.)

We would like to include a large number of tuples from each sequence S into a LMB by setting a "proper" *band-width* Δ, such that the *distances* of outliers from a LMB are *large*. To capture this, we propose a method to *automatically* compute a band-width based on LMBs. For a particular band-width Δ, d_Δ denotes a *distance* of outliers from a LMB and a_Δ denotes a *distinctive degree* of Δ in a sequence of tuples S.

$$a_\Delta = \begin{cases} 0 \text{ if } \Delta = 0; & \frac{d_\Delta - d_{\Delta-1}}{d_\Delta} \text{ otherwise.} \end{cases} \tag{1}$$

For each outlier over a tuple t_j in $S = \{t_1, \cdots, t_j, \cdots, t_n\}$, let $t'_j.\mathbf{Y}$ denote a repair of $t_j.\mathbf{Y}$. If S is a sequence where LMB is a LIB (LDB), then t_{left} denotes the maximal (minimal) tuple in $T[j-1]$ that is part of a LIB (LDB) in S; and t_{right} denotes the minimal (maximal) tuple in $T[j+1,n]$ that is part of a LIB (LDB). We define the estimated repair $t'_j.\mathbf{Y}$ as $(t_{left}.\mathbf{Y} + t_{right}.\mathbf{Y})\ /\ 2$.

Example 3.10: Consider $S = \{t_1 - t_9\}$ in Table I and let $\Delta = 1$. Since the value 2012 of the tuple t_2 over attribute year is incorrect, the repair t'_2.year is calculated as (1992 + 1995) / 2, which is rounded to 1993. □

The distance $d(t, \mathsf{LMB})$ of tuple t from a LMB is computed as $|d(t'.\mathbf{Y}, t.\mathbf{Y})|$. The distance d_Δ of outliers from a LMB is calculated as the average distance i.e., $d_\Delta = \sum_{t \notin \mathsf{LMB}, t \in S} d(t, \mathsf{LMB})\ /\ |\{t : t \notin \mathsf{LMB}, t \in S\}|$.

The band-width Δ is chosen that maximizes the distinctive degree a_Δ. Note that since entire sequence T is divided into contiguous subsequences S, the band-width Δ is the average aggregated value computed over all subsequences S.

Example 3.11: Assume band-width Δ is computed for an attribute year over Table I wrt an OD between cat# and year and an approximation ratio of 0.4 (set higher for traditional ODs as they do not take band-width into account). Hence, the divide-and-conquer method with traditional ODs divides Table I into $T_1 = \{t_1–t_6\}$, $T2 = \{t_7–t_{11}\}$, $T3 = \{t_{12}–t_{17}\}$ and $T4 = \{t_7–t_{11}\}$. Since distinctive degree value is the highest for band-width of 1 wrt T_1, T_2 and T_4 and for band-width of 2 wrt T_3, the averaged band-witdh $\Delta = 1$ (rounded from (1 + 1 + 2 + 1) / 4)). □

IV. Discovery of abcODs

A. Discovery Problem

To make band ODs relevant to real-world applications, we make them less strict to hold *approximately* with some exceptions and *conditionally* on subsets of the data. Given a band OD $\mathbf{X} \mapsto_\Delta \overline{\mathbf{Y}}$, where T is a sequence of tuples ordered by $\mathbf{X}$, our goal is to segment T into multiple contiguous, non-overlapping subsequences of tuples, called *series*, such that (1) large fraction of tuples in each series satisfy a band OD, and (2) outlier tuples that severely violate a band OD in each series are few and sparse. (We experimentally verified in Sec. VI that in practice errors are few and sparse over real-world datasets.)

We define the *approximate band conditional OD (abcOD) discovery problem* as a constrained optimization problem.

Definition 4.1 (abcOD Discovery Problem): Let $\mathbf{X} \mapsto_\Delta \overline{\mathbf{Y}}$ be a band OD, T be a sequence of tuples, ordered by $\mathbf{X}$ over a table r and ε be an approximation error rate parameter. Among all possible non-overlapping segmentations $\mathbb{S}$ of T, the *approximate band conditional OD (abcOD) discovery problem* is to find the optimal segmentation denoted as $\dot{S}$, where $\dot{S} \in \mathbb{S}$ that satisfies the following condition.

$$\max_{\dot{S} \in \mathbb{S}} \quad g(\dot{S}) \qquad \text{s.t. } e(\dot{S}) \leq \varepsilon \tag{2}$$

$g(\dot{S})$ defines the *gain* in terms of portions of $\dot{S}$ satisfying a band OD, and $e(\dot{S})$ defines a *cost* quantifying the number of *errors* in $\dot{S}$ that violate a band OD. For each segment S in $\dot{S}$, let $|S_{nn}|$ be the number of non-null tuples in S, and L_S be a LMB in S. The gain $g(\dot{S})$ and the cost $e(\dot{S})$ are defined respectively as follows.

$$\begin{aligned} g(\dot{S}) = \sum_{S \in \dot{S}} (&|t : t \in L_S, t \in S| \; - \\ &|t : t \notin L_S, t \neq null, t \in S|) \cdot |S_{nn}| \end{aligned} \tag{3}$$

$$e(\dot{S}) = \max_{S \in \dot{S}} e(S) \tag{4}$$

$e(S)$ is the maximum number of contiguous outliers that violate a band OD $\mathbf{X} \mapsto_\Delta \overline{\mathbf{Y}}$ in S.

$$e(S) = \max_{k \in \{i, \cdots, j\}, 1 \leq i \leq j \leq |S|} |j - i| \; : \; t_k \notin L_S, t_k \in S \tag{5}$$

□

Example 4.2: Consider a band OD cat# $\mapsto_{\Delta=1}$ $\overline{\text{year}}$ and an error rate $\varepsilon = 1$. Fig. 2 visualizes three series based on Table I with following abcODs: $\textsf{cat\#} \mapsto_{\Delta=1} \textsf{year}\uparrow$ wrt $S_1 = \{t_1 - t_9\}$, where t_2 is an outlier, $\textsf{cat\#} \mapsto_{\Delta=1} \textsf{year}\downarrow$ wrt $S_2 = \{t_{10} - t_{14}\}$ and $\textsf{cat\#} \mapsto_{\Delta=1} \textsf{year}\uparrow$ wrt $S_3 = \{t_{15} - t_{22}\}$, where t_{18} has a missing year. The segmentation $\dot{S} = \{S_1, S_2, S_3\}$ maximizes the optimization function $g(\dot{S}) = (8-1)\cdot 9 + 5\cdot 5 + 7\cdot 7 = 137$ under a constraint $\max(e(S_1), e(S_2), e(S_3)) = 1 \le \varepsilon$. $\square$

We call band ODs that hold conditionally over subsets of the data and approximately with some exceptions *approximate band conditional ODs* (abcODs). In Equation 3, a gain function rewards correct tuples weighted by the length of series $|S_{nn}|$ excluding tuples with null values to achieve high recall. Otherwise small series would be ranked high.

To identify candidate band ODs without human intervention, we use a global approach to find all traditional ODs within an approximation ratio [6], [7] to narrow the search space, as discovering traditional ODs is less computationally intensive. Since band ODs hold over subsets of the data (with a mix of ascending and descending ordering), we separate an entire sequence of tuples into contiguous subsequences of tuples, by using *divide-and-conquer* approach, such that tuples over contiguous subsequences satisfy a traditional OD within approximation ratio. Found traditional ODs ranked by the measure of interestingness [6], [7] are used as candidate embedded band ODs for the abcODs discovery problem.

Our problem of abcODs discovery is not a simple matter of finding splitting points. We study a technically challenging joint optimization problem motivated by real-life applications of finding splits, monotonic bands and approximation (to account for outliers), which is not easily obtained by simple visualization. Also, note that Figure 1 presents only a small sample of the data extracted from the entire dataset to illustrate the intuition. In practice, the number of data series can be hundreds or thousands over large datasets (see Sec VI), thus, data cannot be split easily into a few partitions. We argue that an automatic approach to discover abcODs is needed as formulating constraints manually requires domain expertise, is prone to human errors, and is excessively time consuming over large datasets.

B. Computing abcODs

We provide an algorithm to compute series for abcODs with an optimal solution denoted as $\textsf{OPT}(n)$, where n denotes a number of tuples over a dataset. The solution to the abcOD discovery problem has an *optimal substructure* property. The optimal solution $\textsf{OPT}(j), j \in \{1, \cdots, n\}$ in a prefix $T[j]$ contains optimal solutions to the subproblems in prefixes $T[1], T[2], \cdots, T[j-1]$.

$$\textsf{OPT}(j) = \begin{cases} 0 & j = 0 \\ \max_{i \in \{1,\cdots,j-1\} \text{ and } e(T[i+1,j]) < \varepsilon}\{ & \\ \quad \textsf{OPT}(i) + g(T[i+1,j])\} & j > 0 \end{cases} \tag{6}$$

We develop a dynamic-programming algorithm (pseudocode in Algorithm 2) to solve the abcOD discovery problem. Two arrays are maintained of the size n: array G stores the overall benefits of optimal solutions to the subproblems, i.e., $G[j] = \textsf{OPT}(j), j \in \{1, \cdots, n\}$; and array X stores the corresponding series, i.e., $X[j]$ stores a segment $\textsf{ID}$ i that tuples $\{t_i – t_j\}$ belong to in a prefix $T[j], 1 \le i \le j \le n$.

Algorithm 2: Computing Series

```
   input  : T = {t_1, t_2, ..., t_n}, Δ, ε
   output : segmentation Ṡ in T
1  X, G ← two arrays of size n+1; Ṡ ← ∅
2  X ← ∅; G ← {0, ..., 0}
3  for j ← [1, n] do
4  |  for i ← [1, j-1] do
5  |  |  L_{i+1,j} ← Compute LMB (T[i+1, j], Δ); E_{i+1,j} ←
   |  |    e(T[i+1, j])
6  |  |  g(T[i+1, j]) ← |L_{i+1,j}|^2 − (|T[i+1, j]| − |L_{i+1,j}|)^2
7  |  |  if E_{i+1,j} ≤ ε and (G[i] + g(T[i+1, j])) > G[j] then
8  |  |  |  G[j] ← G[i] + g(T[i+1, j]); X[j] ← i
9  while j > 1 do add (Ṡ, segmnt (X[j], j, T)); j ← X[j] − 1
10 return Ṡ
```

TABLE III
DYNAMIC PROGRAMMING SAMPLE CALCULATIONS.

	1	2	3	4	5	6	7	8	9	10	11	12	...	22
T	’92	’12	’96	’95	’99	’00	’99	’01	’02	’00	’98	’97	...	’92
G	1	4	5	10	15	24	35	48	63	64	67	72	...	137
X	1	1	2	2	1	1	1	1	1	10	10	10	...	15

The recurrence in Equation 6 specifies that the series in a prefix $T[j]$ are selected among j alternative options: for each $i \in [1, j-1]$, $T[j]$ is split into two sub-sequences: $T[i]$ and $T[i+1, j]$, where series in $T[i]$, with gain $g(T[i])$, is computed by $\textsf{OPT}$(i). We consider $T[i+1, j]$ as a single series and compute its gain $g(T[i+1, j])$ (Line 6). If $g(T[i+1, j]) + g(T[i])$ is greater than existing $\textsf{OPT}$(j), we update $\textsf{OPT}$(j) by $\textsf{OPT}(i) + \textsf{g}(\textsf{T}[i+1], j)$ (Line 8).

Example 4.3: Consider abcOD discovery over records in Table I, given a band OD $\textsf{cat\#} \mapsto_{\Delta=1} \overline{\textsf{year}}$ and an error rate $\varepsilon = 1$. We solve the problem by discovering abcOD in subsequences $T[1]$ till $T[22]$. Prefix $T[1] = \{t_1\}$ is examined first. It forms a singleton series with the benefit $\textsf{OPT}(1) = G[1] = 1$ ($G[1] = 1, X[1] = 1$). We next consider prefix $T[2] = \{t_1, t_2\}$. Tuple t_2 can either form its own series with the benefit equal to 1 (with overall benefit $1 + \textsf{OPT}(1) = 2$) or be merged into the same series with t_1 with the benefit $2^2 = 4$. Thus, t_2 and t_1 are merged as well as $G[2] = 4$ and $X[2] = 1$ are set. The rest of tuples are processed accordingly with the results reported in Table III.

To output all series in $t_1 - t_{22}$, we check array X in reverse order. Given that $X[22] = 15$ stores the minimal tuple $\textsf{ID}$ (15) of a series that $\{t_{15} - t_{22}\}$ belong to, we know that an optimal solution is achieved by a series consisting of $t_{15} - t_{22}$ and an optimal solution in $T[14]$; given $X[14] = 10$, optimal solution in $T[14]$ is achieved by a series $\{t_{10} - t_{14}\}$ and an optimal solution in $T[9]$. Once $X[1] = 1$ is scanned, we find all series shown in Figure 2. $\square$

Theorem 3: Alg. 2 solves the abcOD discovery problem optimally in $O(n^3 \log n)$ time in a sequence T of size n. $\square$

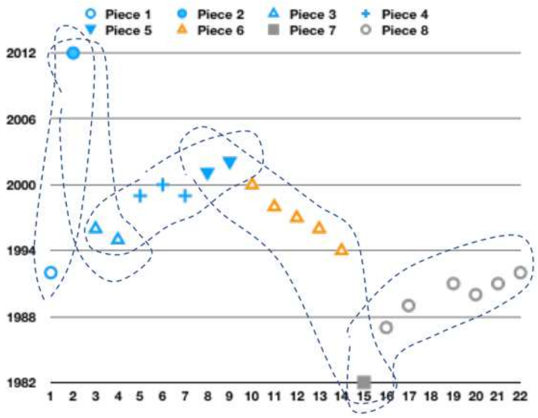


Fig. 5. Pieces and pre-pieces (marked by dash-line).

V. Pruning via Pieces

A. Pieces Decomposition

To further prune the search space, we develop a greedy discovery algorithm that is based on *pieces*. Pieces split a sequence of tuples based on pre-pieces into contiguous subsequences that are monotonic within Δ to speed up the performance without sacrificing the precision (Sec. VI-D).

Definition 5.1 (Pre-Piece): Given a sequence $T = \{t_1, \cdots, t_n\}$ and a list of attributes $\mathbf{Y}$, a contiguous subsequence $T' = \{t_i, t_{i+1}, ..., t_j\}$ is a *pre-piece* (PP) if (1) $\forall_{k,m \in \{i,\cdots,j\}, k<m}$ $t_k \preceq_{\Delta,\overline{\mathbf{Y}}} t_m$ or $t_k.\mathbf{Y}$ = null, and (2) T' cannot be extended without violating the property (1). T' is called an *increasing pre-piece* (IP) if $\overline{\mathbf{Y}} = \mathbf{Y}\uparrow$ and a *decreasing pre-piece* (DP) if $\overline{\mathbf{Y}} = \mathbf{Y}\downarrow$. □

Example 5.2: Let $\Delta = 1$ and $\mathbf{Y} = [\text{year}]$. Consider a sequence of tuples T in Fig. 2 over Table I (ordered by an attribute cat#). There are five pre-pieces in T, i.e., $\{t_1–t_2\}$, $\{t_2–t_4\}$, $\{t_3–t_9\}$, $\{t_8–t_{15}\}$ and $\{t_{15}–t_{22}\}$ as illustrated in Fig. 5. □

A pairwise-disjoint set of *pieces* is obtained by taking the intersections of all pre-pieces.

Definition 5.3 (Piece): Let $T = \{t_1, \cdots, t_n\}$ and $\mathbf{Y}$ be a list of attributes. A *piece* P is a subsequence in T such that: (1) non-overlapping tuples from a pre-piece with other pre-pieces create a separate piece, and (2) overlapping tuples between pre-pieces create a separate piece. □

Example 5.4: Let $\Delta = 1$ and $\mathbf{Y} = [\text{year}]$. Consider a sequence of tuples T in Fig. 2 over Table I (ordered by an attribute cat#). Figure 5 shows eight possible pieces in a sequence T: $P_1 = \{t_1\}, P_2 = \{t_2\}, P_3 = \{t_3, t_4\}, P_4 = \{t_5–t_7\}, P_5 = \{t_8, t_9\}, P_6 = \{t_{10}–t_{14}\}, P_7 = \{t_{15}\}, P_8 = \{t_{16}–t_{22}\}$. □

B. Computing Series with Pieces

To generate pre-pieces, the tuples in T are processed in order. Each tuple $t_j \in T$ is checked, if it can extend IPs and DPs of the maximal length. Otherwise a new IP or DP is generated starting at t_j. To facilitate the process, as shown in Algorithm 3, two maps M_{ins} and M_{dec} are stored. For each tuple $t_j \in T$, when the longest IP is found ending at t_j in a prefix $T[j]$, its length l_{ins} and a maximal tuple max_{tup} are kept as $(\text{max}_{\text{tup}}, l_{\text{inc}})$ in M_{inc} (Line 7). If t_j cannot extend the longest IP ending at t_{j-1}, its starting index is recorded in a sorted array L (Line 9). Similarly, we encode the longest DP that ends at t_j in a prefix $T[j]$ by $(l_{\text{dec}}, \text{min}_{\text{tup}})$ in a map M_{dec}, where min_{tup} is the minimal tuple of the corresponding DP (Line 11–Line 16).

Example 5.5: Let $\Delta = 1$ and $\mathbf{Y} = [\text{year}]$. Consider a sequence of tuples T in Fig. 2 over Table I (ordered by an attribute cat#). When processing t_1, an IP and a DP are found to be ending at t_1 both with the length of one. Next, t_2 is processed, which extends an IP ending at t_1. Thus, the value-pairs are replaced in M_{ins} by $(t_2, 2)$. The rest of the elements in T are processed accordingly. □

Algorithm 3: Compute Pieces

input : $T = \{t_1, t_2, \cdots, t_n\}$, Δ
output : the set of pieces $\dot{P}$ in T
1 $M_{\text{ins}} \leftarrow \emptyset$; $M_{\text{dec}} \leftarrow \emptyset$; $\dot{P} \leftarrow \emptyset$; $L \leftarrow$ array of size n
2 M_{ins}.`updateIfMax` $(t_1, 1)$; M_{des}.`updateIfMin` $(t_1, 1)$
3 **for** ***each*** $j \leftarrow [1, n+1]$ **do**
4   **for** ***each*** (key, value) $\in M_{\text{ins}}$ **do**
5   M_{ins}.`remove` (key), M_{ins}.`updateIfMax` $(t_j, 1)$
6   **if** key $\preceq_{\Delta,\overline{Y}} t_j$ **then**
7   M_{ins}.`updateIfMax` $(\text{max}_{\mathbf{Y}}(t_j, \text{key}), \text{value}+1)$
8   **else if** value $> \max\{M_{\text{dec}}$.`valueSet` $()\}$ **then**
9   L.`insert` $(j - \text{value})$, L.`insert` (j)
10   **for** ***each*** (key, value) $\in M_{\text{dec}}$ **do**
11   M_{dec}.`remove` (key), M_{dec}.`updateIfMin` $(t_j, 1)$
12   **if** key $\preceq_{\Delta,\overline{Y}} t_j$ **then**
13   M_{dec}.`updateIfMin` $(\text{min}_{\mathbf{Y}}(t_j, \text{key}), \text{value}+1)$
14   **else if** value $> \max(M_{\text{ins}}$.`valueSet` $())$ **then**
15   L.`insert` $(j - \text{value})$, L.`insert` (j)
16 $i \leftarrow 0$
17 **while** $L[i+1]$!=null **do**
18   $\dot{P}$.`add` (T.`sub_seq` $(L[i], L[i+1])$)
19   $i \leftarrow i+1$
20 **return** $\dot{P}$

Based on pre-pieces, for non-null tuple pairs $(L[i], L[i+1])$, pieces $P = \{t_{L[i]}–t_{L[i+1]}\}$ are produced separating non-overlapping and overlapping pre-pieces (Line 18).

Lemma 5.6: Algorithm 3 takes $O((\Delta+1) \cdot n)$ time, where n is the size of a sequence of tuples T. □

Our greedy algorithm uses pieces to prune the search space for the abcODs discovery. Instead of processing each tuple individually, we first split the sequence into pieces and then treat tuples in the same piece as a whole.

Theorem 4: The pieces-based algorithm for the abcOD discovery takes $O(m^2 n \log n)$ time, where m is the number of pieces in T, and n is the number of tuples in T. □

In practice, pieces are large, hence, the number of pieces is small (i.e., $m \ll n$).

Example 5.7: Let $\Delta = 1$, $\varepsilon = 1$ and $\mathbf{Y} = [\text{year}]$. Consider a sequence of tuples T in Fig. 2 over Table I (ordered by an attribute cat#). To compute abcODs, Figure 5 illustrates the 8 pieces in T, Table IV includes the information how the benefits are computed and Figure 2 illustrates the series with abcODs over tuples $t_1–t_{22}$. □

TABLE IV

PIECES BASED SAMPLE CALCULATIONS.

	$g(P_0)$	$g(P_1)$	$g(P_{1-2})$	$g(P_{1-3})$	$g(P_{1-4})$	$g(P_{1-5})$	$g(P_{1-6})$	$g(P_{1-7})$
$g(P_1)$	**1**							
$g(P_{1-2})$	**4**	2						
$g(P_{1-3})$	8	**10**	8					
$g(P_{1-4})$	**35**	25	29	19				
$g(P_{1-5})$	**63**	49	53	35	39			
$g(P_{1-6})$	–	–	–	–	84	**88**		
$g(P_{1-7})$	–	–	–	–	**99**	**99**	89	
$g(P_{1-8})$	–	–	–	–	–	–	**137**	135

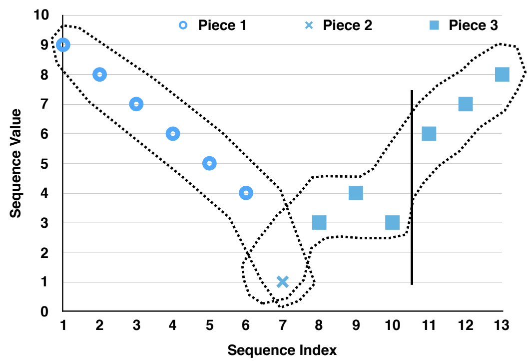


Fig. 6. Pieces-based vs Optimal algorithm.

Interestingly, when only *unidirectional abcODs* (with all ascending or all descending ordering) are considered, the pieces-based algorithm finds the optimal solution in $T[i], i \in [1,n]$. Assume $T[i]$ ends at piece $P_i = \{t_{i-m+1}, \cdots, t_i\}$ of length m. Every tuple in P_i belongs to the same sets of pre-pieces, there are no outliers that violate a LIB in P_i, i.e., $g(T[i-m+1,i]) = m^2$. If the algorithm does not find the optimal solution, then there exists a tuple $t_{i-k} \in P_i, 1 \leq k \leq m-1$ that splits P_i into two series: $\{t_{i-m+1}, \cdots, t_{i-k}\}$ and $\{t_{i-k+1}, \cdots, t_i\}$, where the profit is $\mathsf{OPT}(i-k) + k^2$. By contradiction this assumption does not hold, i.e., $\mathsf{OPT}(i) - \mathsf{OPT}(i-k) \geq k^2$. Let tuple t_{i-j+1} be the first tuple in the last series S_{i-m} of the optimal solution $\mathsf{OPT}(i-m)$, where the length of a LIB in series S_{i-m} is l, i.e., $j \geq m+1, l > 0$; and the maximal number of consecutive outliers in $T[i-m]$ is q. According to Theorem 1, $\{t_{i-m+1}, \cdots, t_i\}$ extends the length of LIB in S_{i-m} by m without increasing q. That is, $\mathsf{OPT}(i) = \mathsf{OPT}(i-j) + (l+m)^2$. Similarly, $\mathsf{OPT}(i-k) = \mathsf{OPT}(i-j) + (l+m-k)^2$. This means that $\mathsf{OPT}(i) - \mathsf{OPT}(i-k) = (l+m)^2 - (l+m-k)^2 = 2k(l+m) > k^2$, which leads to the contradiction.

Theorem 5: The pieces-based algorithm for abcODs discovery finds optimal solution over unidirectional abcODs. □

Over datasets with bidirectional abcODs, the pieces-based approach may produce sub-optimal solutions when adjacent increasing and decreasing pre-pieces are near symmetric with erroneous values on the borders (detailed example in [10])

Example 5.8: Consider a sequence of tuples $T = \{t_1 - t_{13}\}$ in Fig. 6, where an attribute A corresponds to sequence index and an attribute B corresponds to sequence value over an abcOD $\mathsf{A} \mapsto_\Delta \overline{\mathsf{B}}$. Let $\Delta = 1$ and $\varepsilon = 1$. As shown in Fig. 6, there are two pre-pieces in T (denoted with dash lines), and thus, three pieces: $P_1 = \{t_1\text{–}t_6\}, P_2 = \{t_7\}, P_3 = \{t_8\text{–}t_{13}\}$. The pieces-based algorithm finds two solutions in T with the same profit 85: 1) $S_1 = \{t_1\text{–}t_7\}$ and $S_2 = \{t_8\text{–}t_{13}\}$; and 2) $S_1 = \{t_1\text{–}t_6\}$ and $S_2 = \{t_7\text{–}t_{13}\}$. However, the optimal algorithm finds different series: $S_1 = \{t_1\text{–}t_{10}\}$ and $S_2 = \{t_{11}\text{–}t_{13}\}$ with a higher profit 89. □

The above case is very rare in real-world applications (Sec. VI-D).

TABLE V

STATISTICS FOR EXPERIMENTAL DATASETS.

dataset	# tuples	# series	max SS	min SS	avg SS	% MV	% IV
Music-Full	942611	75397	3052	1	12.5	7.8	1
Music-Random	1794	67	433	2	26.9	6.5	1
Music-IncDec	4506	43	433	6	104.8	4.7	2.6
Music-Inc	2188	25	433	6	87.5	3.5	6.0
Music-Simple	376	1	376	376	376	7.4	3.7
Car	362	34	239	1	10.6	8.8	41.7

TABLE VI

SUMMARY OF DISCOVERY METHODS.

	GAP	MONOSCALE	LMS	LMB-OPT	LMB-PIE
Small Violation to Monotonicity		√		√	√
Outliers			√	√	√

VI. EXPERIMENTAL EVALUATION

A. Data Characteristics and Settings

Datasets. We use two real-world datasets for experiments (1) the *Music* dataset (see Footnote 1), and (2) the *Car* dataset (see Footnote 2). The collected *Music* dataset has 1M tuples. It contains information about music releases over 100 years including attributes label, title, country, artists, genres, cat#, format, year and month. The *Car* dataset has 362 tuples. It contains information about second-hand cars including attributes year, vehicle identification number (VIN), orderid, description, model, link, blockid and cartype. We observed that both real-world datasets have missing and incorrect values. Whenever it is not stated otherwise, we report the results with respect to abcODs cat# $\mapsto_\Delta$ $\overline{\text{year}}$ over the *Music* dataset and VIN $\mapsto_\Delta$ $\overline{\text{year}}$ that we automatically identified as candidates for embedded bandODs (as described in Section IV-A).

Real-world Datasets. We categorize the real-world data into five groups by sampling the datasets. Table V shows statistics for the sampled datasets with real-world errors; SS denotes series size, MV missing values and IV incorrect values.

- *Music-Full* is the full *Music* dataset with 1M tuples.
- *Music-Random* is a random sample of the above dataset by providing incomplete information from each series.
- *Music-IncDec* contains series with both ascending and descending trends over abcODs.
- *Music-Inc* contains several music series with only ascending trends (unidirectional abcODs).
- *Music-Simple* all tuples belong to a single series.
- *Car* contains vehicle information from multiple brands.

CER Datasets. Although the real-world *Music* dataset has real errors, we also randomly modify this dataset for some experiments with synthetic errors to control the error rate by either removing or replacing their original values. We denote

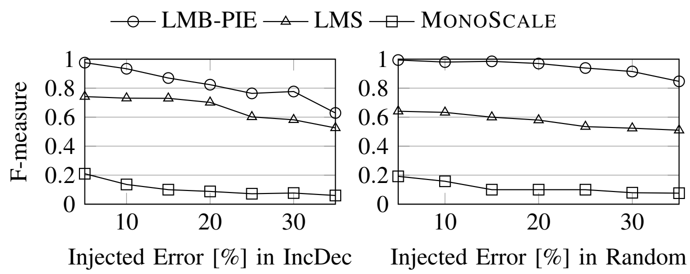


Fig. 7. Discovery quality on *Music CER* datasets.

the perturbed datasets with a controlled error rate as CER datasets. To evaluate the robustness, we vary the missing and erroneous values in the range of 5% to 35%.

Gold Standard. We verify the ground truth as follows.

- *Real-world Datasets*: For all real-world datasets except *Music-Full*, we manually verify the correctness of series wrt abcODs of all variations. For *Music-Full* the values provided are estimates based on our algorithms, and annotations in the original data. This is summarized in Table V.
- *CER datasets*: We use manually-verified ground truth of series wrt abcODs over real-world datasets for CER-datasets.

Algorithms. We developed the following discovery algorithms in Java summarized in Table VI.

- GAP: baseline algorithm that we designed that segments data based on big gaps by outlier detection techniques with 3-standard deviations [11].
- MONOSCALE: discovers series using approximate monotonicity with scale [12]. It tolerates small monotonicity violations, however, does not consider outliers.
- LMS: discovers series by the concept of longest monotonic subsequences (LMS) [2]. It can detect erroneous values in each series, but does not allow small variations.
- LMB-OPT: is our optimal solution (without applying pieces) to discover series.
- LMB-PIE: is our pieces-based solution to discover series.

Our experiments were run on an OS X machine with 2.2 GHz Intel CPU and 16 GB of RAM.

B. Quality of abcOD Discovery

Real-world Data. Table VII presents the results of the abcOD discovery on the real-world datasets. We made the following observations on the *Music* datasets. GAP achieves high recall over all datasets with a large loss in precision. As the algorithm relies on big "gaps" in cat# to discover series and most catalog numbers in the same series are close enough, it only splits series occasionally. Thus, due to its simplicity, the GAP algorithm has a high recall, however, the "gaps" of cat# between consecutive series are not always large, which causes the algorithm to merge series unnecessarily and leads to low precision. MONOSCALE has a high precision and the lowest recall among all algorithms, since it does not take into account outliers and tends to split series, where the outliers occur. To overcome the flaw, we implement a version of the algoirthm called A-MONOSCALE that iteratively removes single outliers to discover series. As shown in Table VII, the adapted method increases recall over all datasets. LMS is tolerant to outliers in each series, however, does not handle small violations to monotonicity (treating them as outliers), hence, it achieves high precision by splitting series (due to many consecutive outliers detected). Finally, our LMB-PIE approach (and thus, also our LMB-OPT approach) overcomes the problems of other techniques. It dominates other approaches over all datasets (F-measure above 0.93 and improved by up to 17% over other methods) and also achieves high accuracy and recall in all datasets (significantly better than other methods). We made similar observations over the results for the *Car* dataset (Table VII) and in Sec. VI-E.

CER Datasets: Fig. 7 illustrates the quality results of abcOD discovery on the *Music* CER datasets. We observe analogous behaviors of all approaches with the controlled error rate as on the real datasets. The algorithm achieves high F-measure (above .82) for a reasonable amount of noise (up to 15%).

C. Band-Width Variations

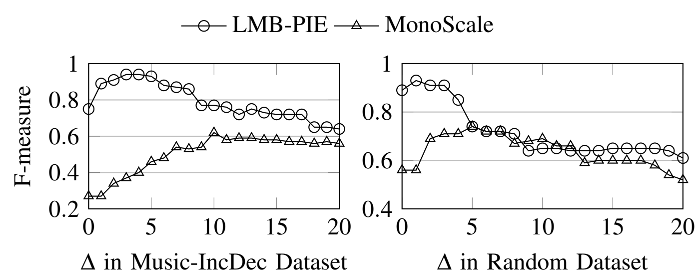


Fig. 8. Discovery when varying band-width Δ.

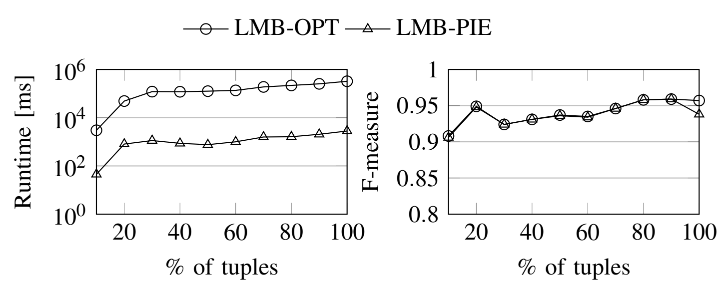


Fig. 9. LMB-OPT vs. LMB-PIE on Music-IncDec dataset.

Note that we manually specify band-width parameter only in this subsection to evaluate the effect of the parameter variations. We compare the results of our abcOD discovery solution on the real-world *Music* datasets with MONOSCALE, where band-width Δ also plays a role. Our solution dominates MONOSCALE in terms of F-measure over the *Music* datasets as reported in Fig. 8. The recall of the algorithm tends to decrease when band-width increases (not shown in Fig. 8). This is because as Δ increases, the method is more tolerant

TABLE VII
DISCOVERY QUALITY ON *Music* AND *Car* DATASETS.

	GAP			MonoScale			A-MonoScale			LMS			LMB-PIE		
	F-1	Precision	Recall	F-1	Precision	Recall	F-1	Precision	Recall	F-1	Precision	Recall	F-1	Precision	Recall
Music-Simple	0.97	1	0.95	0.29	1	0.17	**1**	1	1	**1**	1	1	**1**	1	1
Music-Inc	0.86	0.79	0.95	0.33	0.94	0.20	0.79	0.97	0.67	**0.99**	0.99	0.99	**0.99**	0.99	1
Music-IncDec	0.77	0.63	0.98	0.46	0.83	0.32	0.80	0.91	0.72	0.78	0.98	0.65	**0.95**	0.94	0.95
Music-Random	0.73	0.58	0.99	0.59	0.81	0.47	0.86	0.90	0.82	0.81	0.97	0.69	**0.93**	0.94	0.93
Car	0.53	0.73	0.41	0.35	0.91	0.22				0.96	0.98	0.94	**0.97**	0.98	0.97

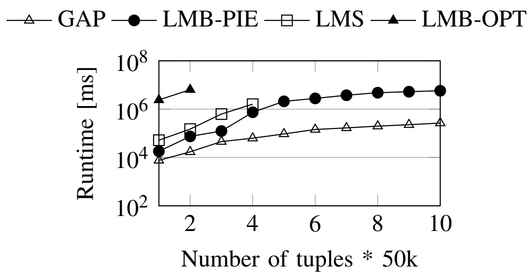


Fig. 10. Runtime on *Music* Dataset.

to monotonicity violations, which leads to wrongly merging series.

Our solution achieves the best F-measure when $\Delta = 3$ on the *Music* dataset. This is because year denotes release date of the records, cat# is assigned to a record at early stages of the production (Section I-A), and the lifespan of producing music records varies from a short period of time to up to a few years based on the complexity of the product and available resources. Our algorithm for automatically discovering the band-width parameter described in Sec III-D finds the right band-width. We also observed that increasing band-width leads to a lower runtime for the abcOD discovery, because as the band-width increases, the LMBs become longer.

D. Efficiency and Effectiveness

We evaluate the scalability of the different discovery algorithms over 500K tuples fraction of the *Music-Full* dataset divided into 10 random portions in Fig. 10. We observe that (1) the pieces-based LMB-PIE algorithm significantly reduces the runtime over the optimal LMB-OPT algorithm on average by two orders-of-magnitude, however, without sacrificing the accuracy as illustrated in Fig. 9 over the *Music-IncDec* dataset divided into 10 portions. The runtime is a consequence of the complexity of the abcOD discovery problem, which for LMB-OPT is super-cubic in the number of tuples (Theorem 3). We developed pruning strategies in the LMB-PIE algorithm that is super-linear in the number of tuples (n) multiplied by quadratic in the number of pieces (p). Note that in practice pieces are large, hence, the number of pieces is small (i.e., $m \ll n$); (2) LMB-PIE has smaller runtime than that of LMS because it generates a smaller number of pieces as LMS does not allow for small variations; (3) MONOSCALE (not shown in Fig. 10) has comparable runtime to LMB-PIE; while GAP is faster than LMB-PIE, due to its simplicity by relying on large gaps, it has a much worse accuracy as reported in Section VI-B.

The LMB-PIE discovery algorithm runs over the *Music-Full* dataset with 1M tuples for around 7h. This is reasonable considering that data profiling is a periodic task. The data discovery engine can be run offline inside the organization, when the resources over the systems are not in use, or when the load is low. This includes nights, and other non-peak hours, such as weekend and holidays. Data dependency discovery is known to be a hard and computationally expensive problem for other types of data dependencies including various variations of functional dependencies [13], [14], [15], order dependencies [3], [6], [7], sequential dependencies [2] and denial constraints [8]. Note that LMB-PIE achieves the same accuracy as LMB-OPT over the *Car* dataset as all series in this dataset are increasing (Theorem 5).

E. Discovery over Multiple Attributes

To measure the effectiveness and efficiency of the abcOD discovery over multiple attributes, we use both the *Music* and *Car* datasets to generate the following data.

- *2-Attributes*: Attribute year is split into centuries and years (e.g., 1993 is 19 and 93).
- *4-Attributes*: Attribute year is split into: millenniums, centuries, decades and years (e.g., 1993 is 1, 9, 9 and 3).

Table VIII shows the results of the abcOD discovery. We observe that our solution over multiple attributes obtains similar F-measure as over a single attribute (i.e., year) in both datasets. Furthermore, our solution over four attributes has lightly lower F-measure, because the distance function leads to slightly different value when year is split into four attributes.

Running abcOD discovery over multiple attributes takes as expected more time (however, not significantly more) than that on a single attribute (Fig 11). We made similar observations by considering other attributes that cannot be computed from year over the *Music* dataset, such as the categorical attribute month, over a band OD cat# $\mapsto_\Delta$ [year, month]$\uparrow$.

F. Candidate Generation

We measured that the divide-and-conquer approach (Section IV-A) based on traditional approximate ODs to identify candidates for embedded bandODs leads to an increased number of reported "errors" without there often being an actual violation. The error rates for the *Music* dataset are 15% and 20% with the abcOD and traditional OD discovery, respectively (the error rates for the *Car* dataset are 15% and 23%, respectively). We verified that embedded band ODs, cat# $\mapsto_\Delta \overline{\text{year}}$ over the *Music* dataset and VIN $\mapsto_\Delta \overline{\text{year}}$ over the *Car* dataset, identified by the global traditional OD discovery by

TABLE VIII
DISCOVERY MULTIPLE ATTRIBUTES ON *Music* AND *Car*.

	1-Attribute			2-Attributes			4-Attributes		
	F-1	Pre.	Recall	F-1	Pre.	Recall	F-1	Pre.	Recall
Simple	1	1	1	1	1	1	1	1	1
Inc	**0.99**	0.99	1	0.98	0.96	0.99	0.91	0.84	1
IncDec	**0.95**	0.94	0.95	0.93	0.90	0.95	0.88	0.80	0.98
Random	**0.93**	0.94	0.93	0.91	0.94	0.88	0.90	0.84	0.97
Car	**0.97**	0.98	0.97	**0.97**	0.98	0.97	0.96	0.98	0.94

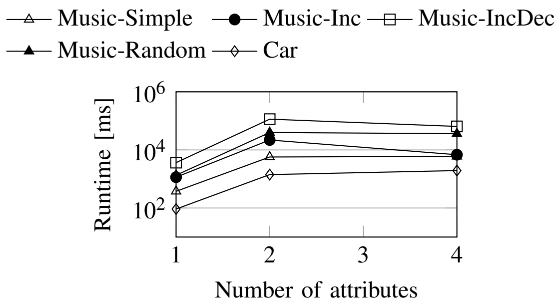


Fig. 11. Runtime *multi-attributes*.

divide-and-conquer method (Sec. IV-A) ranked by the measure of interestingness are indeed the most interesting.

VII. Related Work

Integrity constraints which specify attribute relationships, are commonly used to characterize data quality. Functional dependencies (FDs) are one of the oldest and most popular type of integrity constraints [1]. In practice, dependencies may not hold exactly, due to errors in the data. Thus, approximate FDs have been defined that hold with some exceptions controlled by the number of tuples to be removed from the given table for the dependency to be satisfied. To effectively identify data quality rules different techniques to discover approximate FDs [1], [15], and conditional FDs that hold over subsets of the data, have been developed [13], [14].

A number of extensions to the classical notion of FDs have been proposed to express monotonicity including order dependencies (ODs) [3], [4], [5], [6], [7] that subsume FDs. While discovery of a specified OD can be performed in linear time in the number of tuples, the discovery of a predefined approximate OD raises the complexity to quadratic in the number of attributes for both unidirectional ODs and bidirectional ODs that consider a mix of ascending and descending order [7]. However, the prior work on discovery of approximate ODs [3], [6], [7] does not consider discovering conditional dependencies that hold on subsets of the data (and small variations) as in our work, which is an expensive and involved process. In comparison the complexity of abcOD discovery is super-cubical in the number of tuples for the optimal-algorithm for the bidirectional case (Sec. IV-B) and super-linear in the number of tuples (n), times quadratic in the number of pieces (m), $m \ll n$, for the pieces-based algorithm for abcOD discovery for bidirectional case and for the optimal algorithm for the unidirectional case (Sec. V-B).

Different variations of ODs have been studied including sequential dependencies (SDs) [2]. SDs specify that when tuples have consecutive antecedent values, their consequents must be within a specified proceeding range. The discovery problem was studied for approximate and conditional SDs [2]. However, in contrast to abcODs, SDs do not allow for small variations controlled by band-width, and a mix of ascending and descending trends. Denial Constraints (DCs) [8], a universally quantified first-order logic formalism, are more expressive than ODs [5]. DCs subsume ODs as they allow six operators for comparison of tuples $\{<, >, \leq, \geq, =, \neq\}$. The authors in [8] study the discovery of approximate DCs without considering conditional dependencies. Also, abcODs express order with small variations causing DCs to be violated without actual violation of application semantics.

The abcOD discovery problem is relevant to sequence segmentation [16] into non-overlapping partitions characterized by a model (e.g., mean and median), a general data mining problem for summarizing and analyzing sequential data. Solutions to sequence segmentation fall into two categories [16]: (1) fast heuristic algorithms, including top-down [17], [18], bottom-up [19], [16] and randomized [20] greedy algorithms, and (2) approximation algorithms [16], [21]. Existing sequence segmentation solutions do not consider approximate monotonic segments and allowing for small variations.

VIII. Conclusions

We devise techniques to efficiently and effectively discover a novel data quality rule in the form of abcODs. In future work, we plan to adapt sampling techniques used for functional dependency and key discovery [15] and utilize distributed computing as in previous work on data discovery that includes order operators for ODs [22] to further improve the efficiency of our discovery algorithm. We are also interested in studying inference system for abcODs. the properties of sets of abcODs including axiomatization and inference [4], [6].

References

[1] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "TANE: an efficient algorithm for discovering functional and approximate dependencies," *Comput. J.*, vol. 42, no. 2, pp. 100–111, 1999.

[2] L. Golab, H. Karloff, F. Korn, A. Saha, and D. Srivastava, "Sequential dependencies," *PVLDB*, vol. 2, no. 1, pp. 574–585, 2009.

[3] P. Langer and F. Naumann, "Efficient order dependency detection," *The VLDB Journal*, vol. 25, no. 2, pp. 223–241, 2016.

[4] J. Szlichta, P. Godfrey, and J. Gryz, "Fundamentals of order dependencies," *PVLDB*, vol. 5, no. 11, pp. 1220–1231, 2012.

[5] J. Szlichta, P. Godfrey, J. Gryz, and C. Zuzarte, "Expressiveness and complexity of order dependencies," *PVLDB*, vol. 6, no. 14, pp. 1858–1869, 2013.

[6] J. Szlichta, P. Godfrey, L. Golab, M. Kargar, and D. Srivastava, "Effective and complete discovery of order dependencies via set-based axiomatization," *PVLDB*, vol. 10, no. 7, pp. 721–732, 2017.

[7] J. Szlichta, P. Godfrey, L. Golab, M. Kargar, and D. Srivastava, "Effective and complete discovery of bidirectional order dependencies via set-based axiomatization," *VLDB Journal*, vol. 24, no. 7, pp. 573–591, 2018.

[8] X. Chu, I. Ilyas, and P. Papotti, "Discovering Denial Constraints," *PVLDB*, vol. 6, no. 13, pp. 1498–1509, 2013.

[9] G. Cormode, L. Golab, F. Korn, A. McGregor, D. Srivastava, and X. Zhang, "Estimating the confidence of conditional functional dependencies." in *SIGMOD*, 2009, pp. 469–482.

[10] P. Li, M. Böhlen, D. Srivastava, and J. Szlichta, "Discovery of band order dependencies," Technical Report, 13 pages, available at *https://arxiv.org/abs/1905.11948*, 2019.
[11] V. Barnett and T. Lewis, "Outliers in statistical data," *John Wiley & Sons Ltd., New York*, 1978.
[12] M. Brooks, Y. Yan, and D. Lemire, "Scale-based monotonicity analysis in qualitative modelling with flat segments," in *IJCAI*, 2005, pp. 400–405.
[13] L. Golab, H. Karloff, F. F. Korn, D. Srivastava, and B. Yu, "On generating near-optimal tableaux for conditional functional dependencies," *PVLDB*, vol. 1, no. 1, pp. 376–390, 2008.
[14] W. Fan, F. Geerts, J. Li, and M. Xiong, "Discovering conditional functional dependencies," *TKDE*, vol. 23, no. 5, pp. 683–698, 2011.
[15] T. Papenbrock and F. Naumann, "A Hybrid Approach to Functional Dependency Discovery," in *SIGMOD*, 2016, pp. 821–833.
[16] E. Terzi and P. Tsaparas, "Efficient algorithms for sequence segmentation," in *SIAM*, 2006, pp. 316–327.
[17] V. Lavrenko, M. Schmill, D. Lawrie, P. Ogilvie, D. Jensen, and J. Allan, "Mining of concurrent text and time series," in *SIGKDD*, 2000, pp. 37–44.
[18] P. Wu, S. Carberry, and S. Elzer, "Segmenting line graphs into trends," in *ICAI*, 2010, pp. 697–703.
[19] T. Palpanas, M. Vlachos, E. Keogh, D. Gunopulos, and W. Truppel, "Online amnesic approximation of streaming time series," in *ICDE*, 2004, pp. 338–349.
[20] J. Himberg, K. Korpiaho, H. Mannila, J. Tikanmäki, and H. Toivonen, "Time series segmentation for context recognition in mobile devices," in *ICDM*, 2001, pp. 203–210.
[21] S. Guha, N. Koudas, and K. Shim, "Data-streams and histograms," in *STOC*, 2001, pp. 471–475.
[22] H. Saxena, L. Golab, and I. Ilyas, "Distributed dependency discovery," *PVLDB*, vol. 12, no. 11, pp. 1624–1636, 2019.

Appendix

A. Theorem 1

Proof. Consider the case of IBs in a prefix $T[i-1]$. Tuple $s_{k,i-1}$ is the smallest maximal tuple of $\mathsf{IB}_{k,i-1}$. Since $d(s_{k-1,i-1}.\mathbf{Y}, t_i.\mathbf{Y}) \geq -\Delta$, a tuple t_i is the maximal tuple of a monotonic band $\mathsf{IB}_{k-1,i-1} \cup t_i$ with length k. In addition, $d(t_i.\mathbf{Y}, s_{k-1,i-1}.\mathbf{Y}) \geq 0$, therefore, t_i is the smallest maximal tuple among IBs with length k in a prefix $T[i]$.

Accordingly, consider the case of DBs in a prefix $T[i-1]$. A tuple $l_{k,i-1}$ is the largest minimal tuple of $\mathsf{DB}_{k,i-1}$. As $d(l_{k-1,i-1}.\mathbf{Y}, t_i.\mathbf{Y}) \leq \Delta$, a tuple t_i is the minimal tuple of a monotonic band $\mathsf{DB}_{k-1,i-1} \cup t_i$ with the length k. In addition, $d(t_i, l_{k-1,i-1}) \leq 0$, thus, t_i is a largest minimal tuple among DBs with length k in prefix $T[i]$. $\square$

B. Theorem 2

Proof. To find LMBs in the sequence of tuples T, the key is to find the length of a LMB by identifying the best tuples in T. Since a tuple $B_{\mathsf{inc}}[k_1]$ is updated by the algorithm by $\max(s_{k_1-1}.\mathbf{Y}, t_i.\mathbf{Y})$, where $d(t_i.\mathbf{Y}, s_{k_1-1}.\mathbf{Y}) > 0$, the corresponding band $\mathsf{IB}_{\mathsf{k_1,i}}$ is a IB with smallest maximal tuples that ends at tuple t_i in the sequence $T[i]$. It is also a monotonic band with the shortest length, as k_1 is the smallest index in B_{inc}. Similarly, $\mathsf{IB}_{\mathsf{k_2,i}}$ is a IB of the longest length among IBs with the smallest maximal tuple that ends at t_i in the sequence $T[i]$.

For each $t_i \in T$, we know that the lengths of IBs with the smallest maximal tuples that ends at t_i fall into the range $[k_1, k_2]$. The length of a LIB in $T[i]$ is the maximal value in array $P_{\mathsf{inc}}[i]$. Accordingly, Algorithm 1 finds a LDB with the largest minimal tuple in the sequence T.

For each tuple t_i in the sequence T of length n, it takes time $O(\log n)$ to update array B_{inc}, B_{dec}, P_{inc} and P_{dec}. Therefore, it takes time $O(n \log n)$ to find a LMB in the sequence T. Each tuple t_i inserts maximally $\Delta+1$ values into arrays P_{inc} and P_{dec}; thus, Algorithm 1 takes space $O((\Delta+1)n)$. $\square$

C. Lemma 5.6

Proof. While processing each tuple t_i in the sequence of tuples T of length n two maps M_{inc} and M_{dec} are updated by the algorithm up to $(\Delta+1)$ times, respectively. Therefore, Algorithm 3 takes time $O(\Delta+1) \cdot n$. $\square$

D. Theorem 3

Proof. Algorithm 2 applies dynamic programming to solve Equation 6. The recurrence in Equation 6 specifies that the series in a prefix $T[i]$ are selected among i alternative options: (1) a singleton series consisting of t_i, and the series in a prefix $T[i-1]$; (2) a series of length 2 consisting of $\{t_i, t_{i-1}\}$, and the series in a prefix $T[i-2]$, etc.; and finally, a series of length i consisting of all tuples in a prefix $T[i]$. Therefore, it requires $O(n)$ iterations to find series in a prefix $T[j]$, where each iteration takes time $O(n \log n)$, according to Lemma 2. There are in total n tuples in the sequence T, thus, Algorithm 2 takes time $O(n^3 \log n)$. $\square$

E. Theorem 4

Proof. The proposed piece-based abcOD discovery algorithm first finds all pieces in the sequence T of length n, which takes time $O(\Delta t+1) \cdot n$. Assume the number of pieces is m, the algorithm applies dynamic programming on m pieces, similarly as Algorithm 2, which takes time $O(m^2 n \log n)$. Therefore, the overall time complexity is $O(m^2 n \log n)$. $\square$

F. Theorem 5

Proof. Consider the discovery of unidirectional abcODs case, where without loss of generality all series are increasing; that is, the LMBs in each series are LIBs. We show that the pieces-based algorithm finds the optimal solution in the prefix $T[i]$, which ends at the piece $P_i = \{t_{i-m+1}, \cdots, t_i\}$ of length m.

The last tuple t_i in the prefix $T[i]$ cannot be an outlier of a series in the optimal solution of $T[i]$; otherwise, the profit of solution, where t_i is a singleton series, is always larger, i.e., $\mathsf{OPT}(i) < \mathsf{OPT}(i-1)+1$ according to Equation 6. On the other hand, as every tuple in a piece $P_i = \{t_{i-m+1}, \cdots, t_i\}$ belongs to the same sets of pre-pieces, there is no outliers that violates LIB in P_i; that is, $g(T[i-m+1,i]) = m^2$ and $g(T[i-k+1,i]) = k^2$.

Assume that the piece-based discovery algorithm does not find the optimal solution in $T[i]$; i.e., there exists tuple $t_{i-k} \in P_i, 0 \leq k \leq m-1$ in the optimal solution that splits P_i into two series: $\{t_{i-m+1}, \cdots, t_{i-k}\}$ and $\{t_{i-k+1}, \cdots, t_i\}$, where the profit is $\mathsf{OPT}(i-k)+k^2$. We next prove that this assumption does not hold, i.e., $\mathsf{OPT}(i) - \mathsf{OPT}(i-k) \geq k^2$.

Consider that a tuple t_{i-j+1} is the first tuple in the last series S_{i-m} of the optimal solution $\mathsf{OPT}(i-m)$, where the length of a LIBs in series S_{i-m} is l, i.e., $j \geq m+1, l > 0$; and the maximal number of consecutive outliers in $T[i-m]$ is q. According to Theorem 1, $\{t_{i-m+1}, \cdots, t_i\}$ extends the length of LIB in S_{i-m} by $m-k$ without increasing q. That is

$$\mathsf{OPT}(i) = \mathsf{OPT}(i-j) + (l+m)^2$$

Similarly,

$$\mathsf{OPT}(i-k) = \mathsf{OPT}(i-j) + (l+m-k)^2$$

Which means,

$$\mathsf{OPT}(i) - \mathsf{OPT}(i-k) = (l+m)^2 - (l+m-k)^2 = 2k(l+m) > k^2$$

Thus, the proposed piece-based algorithm finds the optimal solution in the sequence T, where LMBs in all series are increasing. Accordingly, the proposed piece-based algorithm finds the optimal solution in the sequence T, where LMB in all series are decreasing. $\square$